Методичка 7.2 - Применение и обход антиотладочных приемов. Часть 1.

Способы защиты от динамического анализа

Обнаружение виртуальной машины
Выявление признаков виртуального окружения

Обнаружение отладчика
Замер времени выполнения
Контроль целостности кода
Специальные API операционной системы

Если говорить о техниках, которые применяются для защиты программ от динамического анализа, то наиболее часто встречаются проверки, которые замеряют время выполнения команд, осуществляют контроль целостность кода, а также используют специальные функции операционной системы.

На самом деле, способов защиты от динамического анализа очень много, в основном все они направлены на обнаружение отладчика, но также можно встретить и те, которые будут проверять окружение, в котором запускается программа. В основном это сводится к обнаружению косвенных признаков виртуального окружения.

Признаки виртуального окружения

Артефакты в реестре Windows
Имя сетевого интерфейса
MAC-адрес сетевого адаптера

Виртуальную машину можно определить по косвенным признакам, например, такие продукты, как VMware Workstation, VirtualBox всегда оставляют следы в реестре, по которым можно определить факт использования виртуальной машины.

Если мы откроем реестр (regedit) и перейдем в ветку

Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Disk\Enum, то там можно увидеть, что параметр 0 - имеет значение

SCSI\Disk&Ven_VMware_&Prod_VMware_Virtual_S\5&1ec51bf7&0&000000

Как можно заметить, значение параметра содержит такие подстроки, как VMware и Virtual, по этой строке программа может определить, что запуск произошел в виртуальной машине.

Мы, конечно, можем изменить значение этого параметра, но это далеко не единственное место, где встречается подстрока VMware.

Давайте попробуем найти другие значения.

Edit - Find - VMware - F3

Как мы видим, таких мест действительно очень много и после перезагрузки значения восстановятся.

Также можно определить виртуальную машину по имени сетевого интерфейса или по первым трем байтам MAC-адреса.

Откроем командную строку.

> ipconfig /all

...
Physical Address: 00-0C-29-1D-EA-E2
…

В сети Интернет можно найти много сервисов, где по MAC-адресу можно определить производителя сетевого адаптера.

http://standards-oui.ieee.org/oui.txt

```
00-0C-29   (hex)          VMware, Inc.
000C29     (base 16)      VMware, Inc.
                          3401 Hillview Avenue
                          Palo Alto  CA  94304
                          US
```

Как мы видим, производитель данного сетевого адаптера является компания VMware Inc.

Подобный признаков виртуального окружения очень много. Есть интересная утилита, которая называется Paranoid Fish. Данная утилита может также по некоторым критериям проверить, запущены мы в виртуальной среде или нет.

https://github.com/a0rtega/pafish

Запускаем программу pafish.exe и видим, что она нашла еще много признаков, которые говорят о том, что мы работаем в виртуальной машине.

```
[-] VMware detection
[*] Scsi port 0,1,2 ->bus->target id->logical unit id-> 0 identifier ... traced!
[*] Reg key (HKLM\SOFTWARE\VMware, Inc.\VMware Tools) ... traced!
```

[*] Looking for C:\WINDOWS\system32\drivers\vmmouse.sys ... traced!
[*] Looking for C:\WINDOWS\system32\drivers\vmhgfs.sys ... traced!
[*] Looking for a MAC address starting with 00:05:69, 00:0C:29, 00:1C:14 or 00:50:56 ... traced!
[*] Looking for network adapter name ... OK
[*] Looking for pseudo devices ... traced!
[*] Looking for VMware serial number ... traced!

Общие подходы к обходу антиотладочных приемов

Если говорить о способах обхода антиотладочных приемов, то все они сводятся либо к устранению признаков, например, виртуального окружения или работы под отладчиком, либо к поиску места в программе, где реализована проверка и ее отключение.

В разных случаях поступают по-разному. Например, если мы хотим, чтобы программа не смогла определить, что мы работаем на виртуальной машине, мы можем либо убрать все признаки виртуальной машины, что почти невозможно, либо проанализировать программу, найти место в коде, где выполняет проверка и отключить ее, например, методом патчинга.

Способы обнаружения работы программы под отладчиком

Сейчас мы рассмотрим способы обнаружения работы программы под отладчиком.

Достаточно часто используется способ замера времени выполнения команд.

Если программа выполняется в нормальном режиме, то время между двумя командами будет минимальное, а если программа находится под отладчиком и кто-то ее отлаживает по шагам, то время перехода от одной команде к другой очень сильно увеличится. Это явно признак того, что программу отлаживают.

GetTickcount

GetTickCount — функция Windows API, возвращающая количество миллисекунд (1/1000 сек) с момента старта Windows.

Например, функция может использоваться как антиотладочный прием на участке кода, где проверяется ключ. Если проверка ключа длится больше, чем какое-то время, то программа под отладкой.

Инструкция RDTSC

Ассемблерная инструкция для процессоров с архитектурой x86 и x64, читающая счётчик TSC (Time Stamp Counter) и возвращающая в регистрах EDX:EAX 64-битное количество тактов с момента последнего сброса процессора.

Эта инструкция также может быть использована для измерения временных интервалов.

Измерение времени выполнения команд

Для демонстрации примера с измерением времени мы будем использовать функцию GetTickCount().

Исходный код программы prog-5.2.1.c.

```
#include <stdio.h>
#include <windows.h>

int main(int argc, char* argv[]) {

  int diff = 0;
  int startTime = 0;
  int finishTime = 0;

  startTime = GetTickCount();

  for (long k = 0 ; k < 1000000000 ; k++) { }

  finishTime = GetTickCount();

  diff = finishTime - startTime;

  if (diff > 5000) {
    printf("\nDebugger has been detected!\n");
return 0;
  }

  printf("\nResult: %d\n", diff);

  return 0;
}
```

Компилируем программу.

```
> cl prog-5.2.1.c -Od /Gs- /GS- /link /DYNAMICBASE:NO /NXCOMPAT:NO
```

Запускаем без отладчика.

```
>prog-5.1.exe
```

Result: 2485

Запускаем под отладчиком OllyDbg.

Ставим брейкпойнт в начала функции main. Начинаем пошагово выполнять, доходим до цикла и понимаем, что нужно поставить еще один брейкпоинт на выходе из цикла. Ставим и жмем F9. Дальше также пошагово выполняем и видим следующий вывод в программе:

Debugger has been detected (7578)

Времени прошло сильно больше, сработала ловушка и отладчик был обнаружен.

Как можно обойти такую проверку?

Как и было сказано ранее, мы можем либо пропатчить функцию, либо пропатчить программу. В нашем случае функция реализована во внешней библиотеке, а патчить внешнюю библиотеку не самая лучшая идея.

Давайте выполним патчинг нашей программы. Мы видим условный переход JLE и он ведет на нужный нам код.

В нашем случае нужно заменить условный переход на безусловный. Инструкцию JLE на JMP.

Узнаем код инструкции JMP.

http://asm.inightmare.org/opcodelst/index.php?op=JMP

JMP - 0xEB

Binary - Edit

Ставим байт EB.

И идем дальше по шагам. Смотрим вывод программы.

Delta: 7735

Отлично, мы пропатчили программу и обошли проверку.

Стоит отметить, что иногда нужно менять условный переход на инструкции NOP, а не на безусловный переход. Это требуется тогда, когда нужный нам код идет сразу после безусловного перехода.

Поиск отладочных артефактов

Поиск отладочных артефактов также является достаточно эффективным способов для обнаружения отладчика.

Когда программа открыта под отладчиком, то в списке процессов можно найти процесс отладчика по его имени. У отладчика OllyDbg процесс будет называться OLLYDBG.EXE. Мы можем с помощью функции EnumProcess получить список процессов и затем попробовать найти среди них процесс с именем OLLYDBG.EXE. Если такой процесс найден, значит скорее всего программу отлаживают.

Еще один способ это попробовать выполнить поиск окна с именем OllyDbg с помощью функций EnumWindows.

Функция EnumWindows перечисляет все окна верхнего уровня на экране.

Если во множестве всех окон найдено окно с именем “OllyDbg”, то скорее всего программу отлаживают.

Для демонстрации примера с поиском отладочных артефактов мы будем выполнять поиск окна с именем отладчика OllyDbg с помощью функции EnumWindow.

Исходный код программы prog-5.2.2.c.

```
#include <stdio.h>
#include <windows.h>

#pragma comment(lib, "user32.lib")

BOOL CALLBACK findDebugger(HWND hWnd, LPARAM lParam) {

char windowName[64];

   GetWindowText(hWnd, windowName, sizeof(windowName));

if (strstr(windowName, "OllyDbg") != 0) { return FALSE; }

return TRUE;
}

int main(int argc, char* argv[]) {

  if (EnumWindows(findDebugger, 0) == FALSE) {
    printf("\nDebugger has been detected!\n");
```

```
    return 0;
  }

  printf("\nIt's OK!\n");

  return 0;
}
```

Компилируем программу.

> cl prog-5.2.2.c -Od /Gs- /GS- /link /DYNAMICBASE:NO /NXCOMPAT:NO

Запускаем программу:

>prog-5.2.2.exe

It’s OK!

Видим, что программа успешно отработала.

Давайте попробуем запустить программу под отладчиком. Запускаем OllyDbg и открываем нашу программу prog-5.2.2.exe и запускаем ее.

Видим следующее сообщение: Debugger has been detected!

Это значит, среди открытых окон было найдено окно, имя которого содержит подстроку OllyDbg.

Стоит отметить, что для использования этого способа программа должна содержать внутри себя список самых распространенных отладчиков, чтобы выполнять поиск по имени. По этому признаку можно определить, что программа использует подобный антиотладочный прием.

Как можно обойти такую проверку?

Самый простой способ это переименовать отладчик, но это поможет только от способа, который ищет по имени процесса.

В нашем случае это не поможет, потому что имя окна отладчика хранится внутри бинарного файла отладчика.

Первый способ это найти строку OllyDbg в секции данных программы prog-5.2.2 и заменить ее на другую.

Открываем отладчик.

По умолчанию окно Dump указывает на начало секции данных. Мы видим строку OllyDbg. Давайте ее изменим.

003D9000 4F 6C 6C 79 44 62 67 OllyDbg

Давайте заменим первый байт 0x4F на 0x41.

Binary - Edit - 4F - 41.

003D9000 41 6C 6C 79 44 62 67 AllyDbg

Получилась новая строка. Теперь программы будет искать окна, которые будут содержать подстроку AllyDbg. Давайте продолжим выполнение программы - F9.

Видим сообщение - It’s OK!

Это говорит о том, что отладчик не был обнаружен.